文章编号:1001 - 9081(2011)04 - 1057 - 05
doi:10.3724/SP.J.1087.2011.01057

# 基于像素链的直线绘制算法

朱晓林[1],蔡　勇[1],张建生[2]
(1. 西南科技大学 计算机科学与技术学院,四川 绵阳 621010; 2. 西南科技大学 制造科学与工程学院,四川 绵阳 621010)
(linmy24@sina.com)

**摘　要**:针对直线生成算法在直线斜率大于 0.5 时的低效率问题,提出一种基于像素链的直线绘制算法。将直线看做是由许多条平行像素链或对角像素链拼接而成,提出并利用逆向生成直线的类 Bresenham 算法,将斜率在 0.5 ~ 1 的直线绘制转换为斜率在 0 ~ 0.5 的直线绘制,一次判断生成一条像素链。仿真实验表明,基于像素链的算法生成的直线与 Bresenham 算法生成直线一致,且计算量显著减少。该算法只有加法和乘法两种整数运算,适合硬件实现,其绘制速度是 Bresenham 算法的 4 倍。

**关键词**:计算机图形学;直线绘制算法;Bresenham 算法;逆向生成直线;整数运算
**中图分类号**: TP391.41　　**文献标志码**:A

# Line drawing algorithm based on pixel chain

ZHU Xiao-lin[1], CAI Yong[1], ZHANG Jian-sheng[2]
(1. *School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang Sichuan* 621010, *China*;
2. *School of Manufacturing Science and Engineering, Southwest University of Science and Technology, Mianyang Sichuan* 621010, *China*)

**Abstract:** In order to increase the efficiency of the line drawing algorithm when the slope of the line is greater than 0.5, a line drawing algorithm based on pixel chain was proposed. A straight line was treated as an aggregation of several horizontal pixel chains or diagonal pixel chains. An algorithm of line drawing in a reverse direction, which was similar to Bresenham algorithm, was introduced, by which the slope between 0.5 and 1 was converted to that between 0 and 0.5 while generating line. One pixel chain was generated by one judgment. The simulation results show the straight line generated by new algorithm is as same as that generated by Bresenham algorithm, but the calculation is greatly reduced. The new algorithm has generated two integer arithmetic: addition and multiplication, and it is suitable for hardware implementation. The generation speed of the new algorithm is 4 times of Bresenham algorithm with the same complexity of design.

**Key words:** computer graphics; line drawing algorithm; Bresenham algorithm; line generation in reverse direction; integer arithmetic



## 0　引言

直线是最基本的图形元素,是构成复杂图形的基础,直线绘制算法的优劣对整个图形系统的效率和质量有着直接而重要的影响,由于目前人们所采用的显示器多为光栅扫描显示器,所以对直线绘制算法的研究也主要集中在选择距离实际直线最近的光栅点的整数算法上。最著名的直线绘制方法是 20 世纪 60 年代中期出现的 Bresenham 算法[1],该算法中所有的运算都是整数运算,绘制一个点的运算量是 1 次整数加法运算和 1 次符号判断。Bresenham 算法非常适合于硬件实现。1975 年,Gardner 基于直线的对称性提出了新的算法[2],从直线的两端同时生成 2 个点,以提高绘制速度。在 20 世纪 80 年代和 90 年代初期,很多研究者提出了多步算法[3-6]:二步法[3-4]一次判断生成 2 个点,其基本思想是每生成 2 个像素点更新一次决策参数;四步法[5]将直线中可能出现的所有四步行进码事先存储在一个矩阵中,通过判断所绘制直线的形式来确定四步行进码,因此该算法是以增加代码的复杂性和存储容量来提高速度的;多步法[6]提出了更加一般的 8,16,…,$2^n$ 步算法(N-Step),但是该算法的代码复杂性和存储容量更高。21 世纪初,出现了一次有效地特征判断自适应地确定生成的像素数目的算法,这类算法[7-12]一次判断可生成相应像素行上的所有像素点,其效率明显取决于像素行上像素点的数目。对于斜率在[0.5,1]的直线,在大部分像素行上只有一个像素,由于代码的复杂性,其绘制速度并不比 Bresenham 算法快。自适应多步位移码直线绘制算法[13]将直线直接表达成一串由 0 或 1 组成的位移码,通过对直线位移码周期性的分析,给出了一种自适应多步绘制算法,该算法不能精确确定相邻 1 之间 0 的个数以及相邻 0 之间 1 的个数,此外该算法还引入除法运算,不适合硬件实现。自适应多基元直线绘制算法[14]通过对直线离散轨迹的分析,给出了水平位移和对角位移的定义和长度的迭代公式,并以水平位移和对角位移为直线的绘制基元,但该算法在运算中引入了除法运算和求余运算,不适合硬件实现。文献[15]提出了基于对角行程的直线生成算法,将斜率在(0.5,1]的直线分为两类进行判别,提高了直线生成的效率,但增加了算法的冗余。本文通过对 Bresenham 算法的分析,得出了逆向生成直线的类 Bresenham 算法,并进一步提出了将求斜率在(0.5,1]的直线的位移码转化为求斜率在[0,0.5]的直线的位移码的算法。

**收稿日期**:2010 - 10 - 26;**修回日期**:2010 - 11 - 29。　**基金项目**:国家自然科学基金资助项目(10576027)。
**作者简介**:朱晓林(1984 - ),男,山东济宁人,硕士研究生,主要研究方向:计算机图形图像; 蔡勇(1962 - ),男,四川绵阳人,教授,博士,主要研究方向:计算机图形图像处理、虚拟现实; 张建生(1980 - ),男,河北唐山人,讲师,硕士,主要研究方向:逆向工程、计算可视化。

该算法只有加法和乘法运算,适合硬件实现,生成的直线与 Bresenham 算法一致,且能够大大减少生成直线的计算量,提高直线绘制速度。

## 1 直线性质

在数学上,理想的直线是由无限个点组成的集合,考虑到光栅图形输出设备的应用,绘制直线的任务就等价于确定二维像素网格中像素点的位置。不失一般性,本文仅讨论斜率在[0,1]的直线。

为方便讨论,设直线的起点为(0,0),终点为$(x_0,y_0)$,$x_0 \geqslant 0, y_0 \geqslant 0$,而对一般的情况则利用平移变换容易求得。记 $dx = x_0, dy = y_0$,决策参数为 $D$,决策参数增量分别为 $incrD_1$ 和 $incrD_2$;将斜率在[0,1]的直线进一步分为两大类,即斜率在[0,0.5]和在(0.5,1]的直线。

结论 1 Bresenham 算法生成的直线在除起始和终止两像素行外,其他各像素行的像素点数个数 $e$ 满足:

$$Q \leqslant e \leqslant Q + 1 \tag{1}$$

式中 $Q$ 为$(dx/dy)$的下取整("/"表示除法)(文献[11]已证明)。

例如,一条从(0,0)到(50,9)的直线,其 $Q = 5$,则该直线除起始和终止两像素行外,其他各像素行的像素点个数为 5 或 6。

当直线斜率小于 0.5 时,一次判断确定生成像素点数目的方法[11]为:在生成一条直线时,只要在两种情况中作出选择就可以一次性生成一个像素行。设一个像素行的第一像素点决策参数为 $D_0$,考查该点起第 $Q$ 个像素点的决策参数:

$$D = D_0 + 2dy \times (Q - 1) \tag{2}$$

若 $D < 0$,则该行有 $e = Q + 1$ 个像素点,决策参数增量为:

$$line\_incrD_2 = 2dy \times Q - 2dx \tag{3}$$

否则 $D \geqslant 0$,该行有 $e = Q$ 个像素点,决策参数增量为:

$$line\_incrD_1 = 2dy \times (Q - 1) - 2dx \tag{4}$$

这样通过一次比较就可以确定一个像素行的像素点数目。根据该行像素点的数目 $e$ 对 $D$ 予以增量:

$$D = D_0 + 2dy \times e - 2dx \tag{5}$$

为下一像素行的计算作好准备。

$Q$ 值的计算是关键点,本文利用第二行像素点个数来计算。

**定理 1** 若 Bresenham 算法生成的直线的第 2 个像素行(即 $y = 1$ 的像素行)有 $n$ 个像素点,若 $dy \times n \leqslant dx$,有:

$$Q = n \tag{6}$$

否则有: $$Q = n - 1 \tag{7}$$

证明 由式(1)推出 $dy \times n \leqslant dx$ 且 $dy \times (Q+1) \geqslant dx$:

1) 若 $Q = n + 1$,推出 $n \leqslant Q$,与结论 1 矛盾。

2) 若 $Q = n - 2$,推出 $Q + 1 \leqslant n$,与结论 1 矛盾。

由 1)、2) 可以确定 $n - 1 \leqslant Q \leqslant n$。

3) 若 $dy \times n \leqslant dx$ 且 $Q = n - 1$ 时,与 $dy \times (Q+1) \geqslant dx$ 矛盾。故当 $dy \times n \leqslant dx$ 时,有 $Q = n$。

4) 若 $dy \times n > dx$ 且 $Q = n$ 时,与 $dy \times n \leqslant dx$ 矛盾。故当 $dy \times n > dx$ 时,有 $Q = n - 1$。

结论得证,只需一次判断即可确定 $Q$ 值。

在利用 Bresenham 算法生成直线时,主要关注的是纵向坐标 $y$ 的变化。图 1 中,在直线生成过程中横向坐标 $x$ 的增量为 1,纵向坐标要么为 0,要么为 1,这些值构成直线的位移码[13]。一串长度为 $len$ 的"0"位移码,也称为长度为 $len$ 的水平像素链;一串长度为 $len$ 的"1"位移码表示长度为 $len$ 的对角像素链。这类编码可对直线进行精确描述,为探索直线之间的关系提供了一种方法。

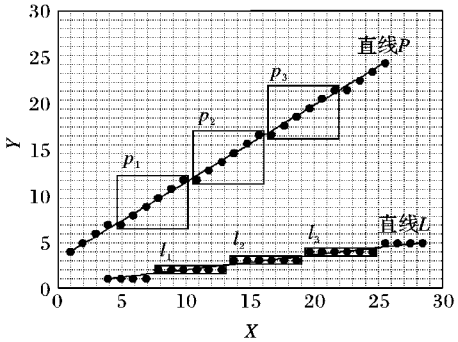


图 1 Bresenham 算法生成直线图例

## 2 直线逆向生成的类 Bresenham 算法

在应用 Bresenham 算法生成直线时,生成方向是正向的,即 $x$ 由左向右来生成直线。根据 Bresenham 算法,可以推导出逆向的类 Bresenham 直线生成算法,即 $x$ 由右向左来生成直线。为了叙述方便,逆向算法暂命名为 Reverse 算法。

以从点(0,0),到点(10,4)的直线为例,如图 2。由 Bresenham 算法原理,由点(0,0)开始正向判断,目标直线与直线 $x = 1$ 的交点为 $P$,若从点(0,0)到点 $P$ 在 $Y$ 方向的增量大于或等于 0.5,则下一个像素点在右上方,记位移码为 1;否则取右方向上的点,记位移码为 0。

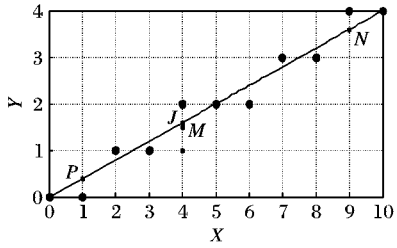


图 2 Reverse 算法图例

反过来,若从终点(10,4)开始向左逆向判断,目标直线与直线 $x = 9$ 交点为 $N$,若从点(10,4)到点 $N$ 在 $-Y$ 方向上的增量小于等于 0.5,为保证与 Bresenham 算法生成的直线一致,下一个像素点应在左方向上,记位移码为 0;否则取左下方向上的点,记位移码为 1。

以点(3,1)正方向判断和以点(5,2)逆向判断进行对比:目标直线与直线 $x = 4$ 交点为点 $J$,点 $M$ 坐标为(4,1.5)。由 Bresenham 算法,若点 $J$ 在 $M$ 点上方或者与 $M$ 点重合,即从点(3,1)到点 $P$ 在 $Y$ 方向的增量大于或等于 0.5,则下一个像素点取点(4,2),否则取点(4,1)。而由 Reverse 算法,若点 $J$ 在 $M$ 点上方或者与 $M$ 点重合,即从判断点到点 $J$ 在 $-Y$ 方向上增量小于等于 0.5,下一像素点取右方向上的点(4,2)。这体现了两种算法在生成精度上的一致性。表 1 给出两种算法的具体对比。同样,通过一次有效地特征判断自适应地确定生成的像素数目的直线生成算法方法,也适用于 Reverse 算法。记 Bresenham 算法生成的位移码为 $B(x,y)$,Reverse 算法生成的位移码为 $B^{-1}(x,y)$。

表 1　Bresenham 算法与 Reverse 算法的对比

| 直线绘制算法 | 判别方向 | 决策参数初值 | $D$ 取值范围 | 取点方向、位移码 | 决策参数递推关系 |
|---|---|---|---|---|---|
| Bresenham 算法 | 正方向 | $D_0 = 2dy - dx$ | $D \geqslant 0$ | 位移码为 1,取右上方点 | $D = D + 2dy - 2dx$ |
| | | | $D < 0$ | 位移码为 0,取右方向点 | $D = D + 2dy$ |
| Reverse 算法 | 逆方向 | $D_0 = 2dy - dx$ | $D > 0$ | 位移码为 1,取左下方点 | $D = D + 2dy - 2dx$ |
| | | | $D \leqslant 0$ | 位移码为 0,取左方向点 | $D = D + 2dy$ |

## 3　基于像素链的直线生成算法

对于第 1 类直线,即斜率在[0,0.5]上的直线,用 Bresenham 算法生成。

1)初始化决策参数:$D_0 = 2dy - dx$,决策参数增量:$incrD_1 = 2dy - 2dx$,$incrD_2 = 2dy$;直接生成前两像素行,记录第二行像素点个数 $n$ 并求得 $Q$。

2) 像素行的决策参数:$D = D' + 2dy \times (Q - 1)$,$D'$ 为该像素行第一个像素点的决策参数,当 $D < 0$ 时,该行有 $Q$ 个像素点,否则该行有 $Q + 1$ 个像素点;$line\_incrD_1 = 2dy \times (Q - 1) - 2dx$,$line\_incrD_2 = 2dy \times Q - 2dx$;分别为该行有 $Q$ 和 $Q + 1$ 个像素点对应的决策参数增量。用这种一次有效特征判断确定生成的像素数目的方法生成第三行至倒数第二行像素点。

3) 生成最后一个像素行。

对于斜率在(0.5,1]之间的第 2 类直线,其位移码的生成可以转换为用 Reverse 算法求第一类直线的位移码。

**定理 2**　起止点为 $(0,0)$、$(x_0,y_0)$ 的直线 $P$ 属于第 2 类直线;起止点为 $(0,0)$、$(x_0, x_0 - y_0)$ 的直线 $L$ 属于第 1 类直线,Bresenham 算法生成直线 $P$ 的位移码记为 $B(x_0,y_0)$,Reverse 算法生成直线 $L$ 的位移码为 $B^{-1}(x_0, x_0 - y_0)$,则有:

$$B(x_0,y_0) + B^{-1}(x_0,x_0 - y_0) = B(x_0,x_0) = \underbrace{11\cdots111}_{x_0\text{个}} \tag{8}$$

证明　Bresenham 算法求直线的位移码,$dx\_P = x_0 = dx$,$dy\_P = y_0 = dy$;直线 $L$ 的逆向位移码用 Reverse 算法 $dx\_L = x_0 = dx$,$dy\_L = x_0 - y_0 = dx - dy$。

表 2 是生成两条直线的具体算法比较。

表 2　直线 $P$ 和直线 $L$ 的算法比较

| 直线绘制算法 | 决策参数初值 | $D\_P/D\_L$ 取值范围 | 位移码 | 决策参数递推关系 |
|---|---|---|---|---|
| Bresenham 算法生成直线 $P$ | $D_0\_P = 2dy - dx$ | $D\_P \geqslant 0$ | 位移码为 1 | $D\_P = D\_P + 2dy - 2dx$ |
| | | $D\_P < 0$ | 位移码为 0 | $D\_P = D\_P + 2dy$ |
| Reverse 算法生成直线 $L$ | $D_0\_L = dx - 2dy$ | $D\_L > 0$ | 位移码为 1 | $D\_L = D\_L - 2dy$ |
| | | $D\_L \leqslant 0$ | 位移码为 0 | $D\_L = D\_L - 2dy + 2dx$ |

决策参数初值之和:

$$D\_P + D\_L = 0 \Rightarrow \begin{cases} D\_P \geqslant 0, & D\_L \leqslant 0 \\ D\_P < 0, & D\_L > 0 \end{cases} \tag{9}$$

一方面,$D\_P \geqslant 0$,$D\_L \leqslant 0$,有:

$$\begin{cases} D\_P = D\_P + 2dy - 2dx, & B(P) = 1 \\ D\_L = D\_L - 2dy + 2dx, & B^{-1}(L) = 0 \end{cases} \tag{10}$$

可推出
$$\begin{cases} D\_P + D\_L = 0 \\ B(P) + B^{-1}(L) = 1 \end{cases} \tag{11}$$

另一方面,$D\_P < 0$,$D\_L > 0$,有:

$$\begin{cases} D\_P = D\_P + 2dy, & B(P) = 0 \\ D\_L + D\_L - 2dy, & B^{-1}(L) = 1 \end{cases} \tag{12}$$

可推出
$$\begin{cases} D\_P + D\_L = 0 \\ B(P) + B^{-1}(L) = 1 \end{cases} \tag{13}$$

综合以上分析得出两直线编码的互补关系,即只存在两种位移码形式:

$$\begin{cases} B(P) = 1 \\ B^{-1}(L) = 0 \end{cases} \tag{14}$$

和
$$\begin{cases} B(P) = 0 \\ B^{-1}(L) = 1 \end{cases} \tag{15}$$

无论哪种编码形式都有:

$$\begin{cases} B(P) + B^{-1}(L) = 1 \\ D\_P + D\_L = 0 \end{cases} \tag{16}$$

故定理得证。

以从点(0,0)到点(20,15)的直线 $P$ 为例,如图 3;与之相关是点(0,0)到点(20,5)的直线 $L$,如图 4。

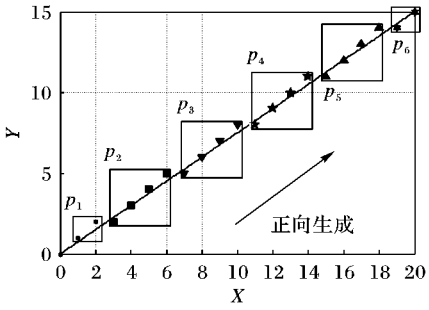


图 3　直线 $P$ 的编码示意图

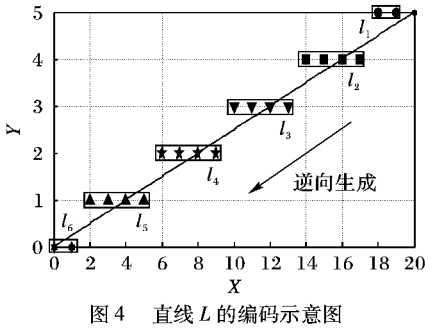


图 4　直线 $L$ 的编码示意图

下面是直线 $P$ 的位移码 $B(20,15)$ 与直线 $L$ 的位移码 $B^{-1}(20,5)$ 之间的对比:

$$\overbrace{11}^{p_1}\ \overbrace{0111}^{p_2}\ \overbrace{0111}^{p_3}\ \overbrace{0111}^{p_4}\ \overbrace{0111}^{p_5}\ \overbrace{01}^{p_6}$$

$$\underbrace{00}_{l_1}\ \underbrace{1000}_{l_2}\ \underbrace{1000}_{l_3}\ \underbrace{1000}_{l_4}\ \underbrace{1000}_{l_5}\ \underbrace{10}_{l_6}$$

图3中$p_i$像素段编码对应图4中的$l_j$像素段编码，显然满足定理。

由定理2，求第2类直线的位移码可转化为求对应的第1类直线的位移码。综合第1类直线的生成原理和定理2，可得基于像素链的直线绘制算法。

算法1　基于像素链的直线绘制算法。

```
x←0; y←0; n←0;
If (2y0 ≤ x0)                                    //第一类直线
{ mode←0;dx←x0; dy←y0;D←2dy - dx;
  incrD1←2dy - 2dx; incrD2←2dy;
  While (y ≤ 1){Draw(x, y);
     If (D < 0){D←D + incrD2;}
     Else {y←y + 1; D←D + incrD1;}
     x←x + 1;
     If (y = = 1){n = n + 1;}}
  If (n * dy > dx){Q←n - 1}
  Else {Q←n;}
  Line_incrD1←- 2dx;
  Line_incrD2←2dy - 2dx;
  While (y < dy){
     D←D + 2dy * (Q - 1);
     If (D < 0){D←D + Line_incrD2;
       Draw_line(x,y,Q + 1,mode);
       x←x + Q + 1;}
     Else {D←D + Line_incrD1;
       Draw_line(x,y,Q + 1,mode);
       x←x + Q;}
     y←y + 1;}
  Draw_line(x,y,dx - x + 1,mode);}
Else                                             //第二类直线
  { mode←1; dx←x0; dy←x0 - y0;
    temp_x←dx;temp_y←dy; D←2dy - dx;
    incrD1←2dy - 2dx; incrD2←2dy;
    While (temp_y ≥ dy - 1){ Draw(x,y);
       If (D ≤ 0){ D←D + incrD2; y←y + 1;}
    Else{ temp_y←temp_y - 1;
      D←D + incrD1;}
    x←x + 1; temp_x←temp_x - 1;
      If (temp_y = = dy - 1){ n←n + 1;}}
  If (n * dy > dx){ Q←n - 1;}
  Else { Q←n;}
  Line_incrD1←- 2dx;
  Line_incrD2←2dy - 2dx;
  While (y < dy){
     D←D + 2dy * (Q - 1);
     If (D ≤ 0){D←D + Line_incrD2;
       Draw_line(x,y,Q + 1,mode);
       x←x + Q + 1; y←y + Q;
       temp_x←temp_x - Q - 1;}
     Else { D←D + Line_incrD1;
       Draw_line(x,y,Q,mode);
       x←x + Q; y = y + Q - 1;
       temp_x←temp_x - Q;}
     temp_y←temp_y - 1;}
  Draw_line(x,y,dx - x + 1,mode);}
```

其中：$(x_0,y_0)$为输入，表示生成目标直线的终点坐标，起点默认为(0,0)；$n$为第一个像素行中像素点的个数；$temp_x$与$temp_y$为计算第二类直线位移码所需的坐标；子程序Draw_line($x$,$y$,$len$,$mode$)是指当$mode$为0代表一串长度为$len$的“0”位移码，即点亮起点坐标为($x$,$y$)长度为$len$的水平像素链；否则$mode$ = 1代表一串长度为$len$的“1”位移码，即点亮起点坐标为($x$,$y$)长度为$len$的对角像素链。子程序Draw($x$,$y$)点亮坐标为($x$,$y$)的像素点。

## 4　实验结果分析与比较

考虑到实际应用，我们用计算量分析的方法对Bresenham算法、改进的Bresenham算法（文献[11]算法）及基于像素链的算法进行比较。对于起点为($x_1$,$x_2$)终点为($y_1$,$y_2$)的直线，令$dx = x_2 - x_1$，$dy = y_2 - y_1$，直线绘制算法的计算量仅与$dx$和$dy$的大小有关，而与起点和终点无关，因此，绘制起点为(0,0)终点($dx$,$dy$)的直线与绘制起点为($x_1$,$x_2$)终点为($y_1$,$y_2$)的直线有着相同的计算量。为方便讨论，可用($dx$,$dy$)来表示直线。表3给出三种算法的计算量比较。

**表3　算法计算量比较**

| 直线绘制算法 | 斜率 | 加减法 | 符号位判断 |
|---|---|---|---|
| Bresenham算法 | $0 \leqslant k \leqslant 1$ | $dx$ | $dx$ |
| 改进的Bresenham算法 | $0 \leqslant k \leqslant 1$ | $3dx/2dy + dy$ | $3dx/2dy + dy$ |
| 基于像素链的算法 | $0 \leqslant k \leqslant 0.5$ | $3dx/2dy + dy$ | $3dx/2dy + dy$ |
| | $0.5 < k \leqslant 1$ | $3dx/2(dx - dy) + dx - dy$ | $3dx/2(dx - dy) + dx - dy$ |

该算法已在计算机上得到实现，其生成的像素点的坐标与Bresenham算法一致，且执行效率明显优于传统的Bresenham算法。图5是基于像素链的算法与Bresenham算法及改进的Bresenham算法在分别编程实现后的效率比较图，该图是在Intel Core2 Quad CPU Q9400 2.66 GHz，3.25 GB内存的计算机上用Matlab 7.0计算所得。目标直线的起点为(0,0)，为方便比较，终止点的$X$坐标定为100 000，即直线长度为100 000。图中横坐标为目标直线终止点的$Y$坐标，纵坐标为计算直线位移码的时间，不包括像素点点亮时间。

由图5比较可见，在生成斜率在[0,0.5]上的直线时，基于像素链的算法与改进的Bresenham算法相比，生成效率大致相同。在生成斜率在(0.5,1]上的直线时，充分体现了新算法的优点，执行效率也比改进的Bresenham算法有了较大幅度的提高。

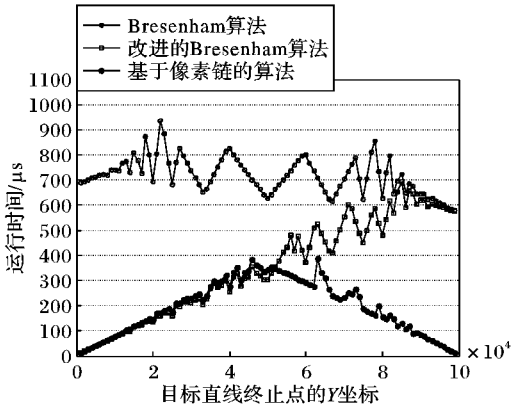


图5　算法的效率比较

## 5　结语

本文通过对 Bresenham 算法的分析,得出了逆向生成直线的类 Bresenham 算法,并进一步提出了将求斜率在(0.5,1]上的直线的位移码转化为求斜率在[0,0.5]的直线的位移码的算法。该算法只有加法和乘法运算,适合硬件实现,生成的直线与 Bresenham 算法一致,且能够大大减少生成直线的计算量,提高直线绘制速度。

在第 1 类直线中的 $y_0 = 1$ 和第 2 类直线中的 $x_0 - y_0 = 1$ 时,基于像素链的算法会退化为单步的 Bresenham 算法。在未来的工作中,我们将继续完善直线绘制算法。

**参考文献:**


[1] BRESENHAM J E. Algorithm for computer control of a digital plotter[J]. IBM Systems Journal, 1965, 4(1): 25 – 30.

[2] GARDNER P L. Modifications of Bresenham's algorithm for display [J]. IBM Technical Disclosure Bulletin, 1975, 18(5): 1595 – 1596.

[3] WU X, ROKNE J G. Double-step incremental generation of lines and circles[J]. Computer Vision, Graphics, and Image Processing, 1987, 37(3): 331 – 344.

[4] GRAHAM P, IYENGAR S S. Double and triple-step incremental linear interpolation[J]. IEEE Computer Graphics and Applications, 1994, 14(3): 49 – 53.

[5] BAO P G, ROKNE J G. Quadruple-step line generation[J]. Computers & Graphics, 1989, 13(4): 461 – 469.

[6] GILL G W. N-step incremental straight-line algorithms[J]. IEEE Computer Graphics and Applications, 1994, 14(3): 66 – 72.

[7] BRESENHAM J E, GRICE D G, PI S C. Run length slice algorithms for incremental lines[J]. IBM Technical Disclosure Bulletin, 1980, 22(8B): 3744 – 3747.

[8] BOYER V, BOURDIN J J. Auto-adaptive step straight-line algorithm[J]. IEEE Computer Graphics and Applications, 2000, 20(5): 67 – 69.

[9] 郑宏珍, 赵恢. 改进的 Bresenham 直线生成算法[J]. 中国图象图形学报, 1999, 4(7): 606 – 609.

[10] 刘晶. 改进的 Bresenham 直线生成算法[J]. 计算机应用与软件, 2008, 25(10): 247 – 249.

[11] 贾银亮. Bresenham 直线生成算法的改进[J]. 中国图象图形学报, 2008, 1(13): 158 – 161.

[12] 刘晶. 快速直线生成算法[J]. 金陵科技学院学报, 2007, 23(3): 9 – 12.

[13] 苗兰芳. 自适应多步位移码直线绘制算法[J]. 软件学报, 2002, 13(4): 637 – 642.

[14] 蔺想红, 张田文. 自适应多基元直线绘制算法[J]. 计算机辅助设计与图形学学报, 2006, 18(8): 1136 – 1141.

[15] 叶晓彤, 邓云. 基于对角行程的直线生成算法研究[J]. 计算机应用, 2008, 28(9): 2270 – 2273.


(上接第 1049 页)

Camshift 算法、Camshift 算法结合卡尔曼预测(CK)和本文方法(BhaCK)对球体运动视频序列的跟踪精度对比结果,可以看到本文方法具有明显优势。

表 1　跟踪精度比较

| 算法 | 跟踪精度 |
|---|---|
| Camshit | 21.8871 |
| CK | 17.2559 |
| BhaCK | 3.7629 |

## 5　结语

本文提出一种从时域卡尔曼预测到空域 Camshift 精确匹配的视频目标跟踪方法。当运动目标机动性比较强时卡尔曼滤波器建立的运动模型与目标真实的运动模型不一致,因此导致单步预测误差值较大,容易使下一步 Camshift 跟踪失败。采用一种在卡尔曼预测窗口基础上适当扩大搜索范围然后再用 Bhattacharyya 系数核匹配方法确定初始匹配窗口,在这个窗口中再用 Camshift 算法精确跟踪视频目标的方法。通过实验表明本文方法能够有效地定位运动目标在每一帧的初始位置,解决了目标前后帧空间上不存在重叠的跟踪难点,取得了令人满意的跟踪精度。

**参考文献:**


[1] ZHOU S K, CHELLAPPA R, MOGHADDAM B. Visual tracking and recognition using appearance-adaptive models in particle filters [J]. IEEE Transactions on Image Processing, 2004, 13(11): 1491 – 1506.

[2] 虞旦, 韦巍, 张远辉. 一种基于卡尔曼预测的动态目标跟踪算法研究[J]. 光电工程, 2009, 36(1): 52 – 56.

[3] ZHE DONG, ZHENG YOU. A novel extended Kalman filter for a class of nonlinear systems [J]. Progress in Natural Science, 2006, 16(9): 912 – 918.

[4] JULIER S J, UHLMANN J K. Unscented filtering and nonlinear estimation [J]. Proceedings of the IEEE, 2004, 92(3): 401 – 422.

[5] COMANICIU D, MEER P. Mean shift: A robust application toward feature space analysis[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(5): 603 – 619.

[6] 朱胜利, 朱善安, 李旭超. 快速运动目标的 Mean shift 跟踪算法 [J]. 中国图象图形学报, 2005, 33(5): 66 – 70.

[7] SOHAIL K M, UMAR I M, SAQUID S M, *et al.* Bhattacharyya coefficient in correlation of gray-scale objects[J]. Journal of Multimedia, 2006, 1(1): 56 – 61.

[8] 卢璇, 雷航, 赫宗波. 联合多特征的自动 CamShift 跟踪算法 [J]. 计算机应用, 2010, 30(3): 650 – 652.

[9] 胡波, 陈恳, 徐建瑜, 等. 基于 Kalman 预测和 Mean-shift 算法的视频目标跟踪[J]. 光电子·激光, 2009, 20(11): 1517 – 1522.

[10] WENG S-K, KUO C-M, TU S-K. Video object tracking using adaptive Kalman filter[J]. Journal of Visual Communication and Image Representation, 2006, 17(6): 1196 – 1197.

[11] WANG ZHAO-WEN, YANG XIAO-KANG, XU YI, *et al.* Camshift guided particle filter for visual tracking[J]. Pattern Recognition Letters, 2009, 30(4): 407 – 413.

[12] 赵祎, 穆志纯, 刘克. 基于肤色及轮廓信息的人耳实时跟踪 [J]. 中国图象图形学报, 2006, 11(7): 949 – 953.

[13] NUMMIARO K, KOLLER-MEIER E, GOOL V L. A color-based particle filter [C]// Proceedings of 1st International Workshop on Generative-Model-Based Vision. Copenhagen: European Conference on Computer Vision, 2002: 53 – 60.